Function and Recursion Euclid’s Greatest Common Divisor(GCD) Algorithm

Week-01, Lecture-02

Course Code: CSE221

Course Title: Algorithms

Program: B.Sc. in CSE

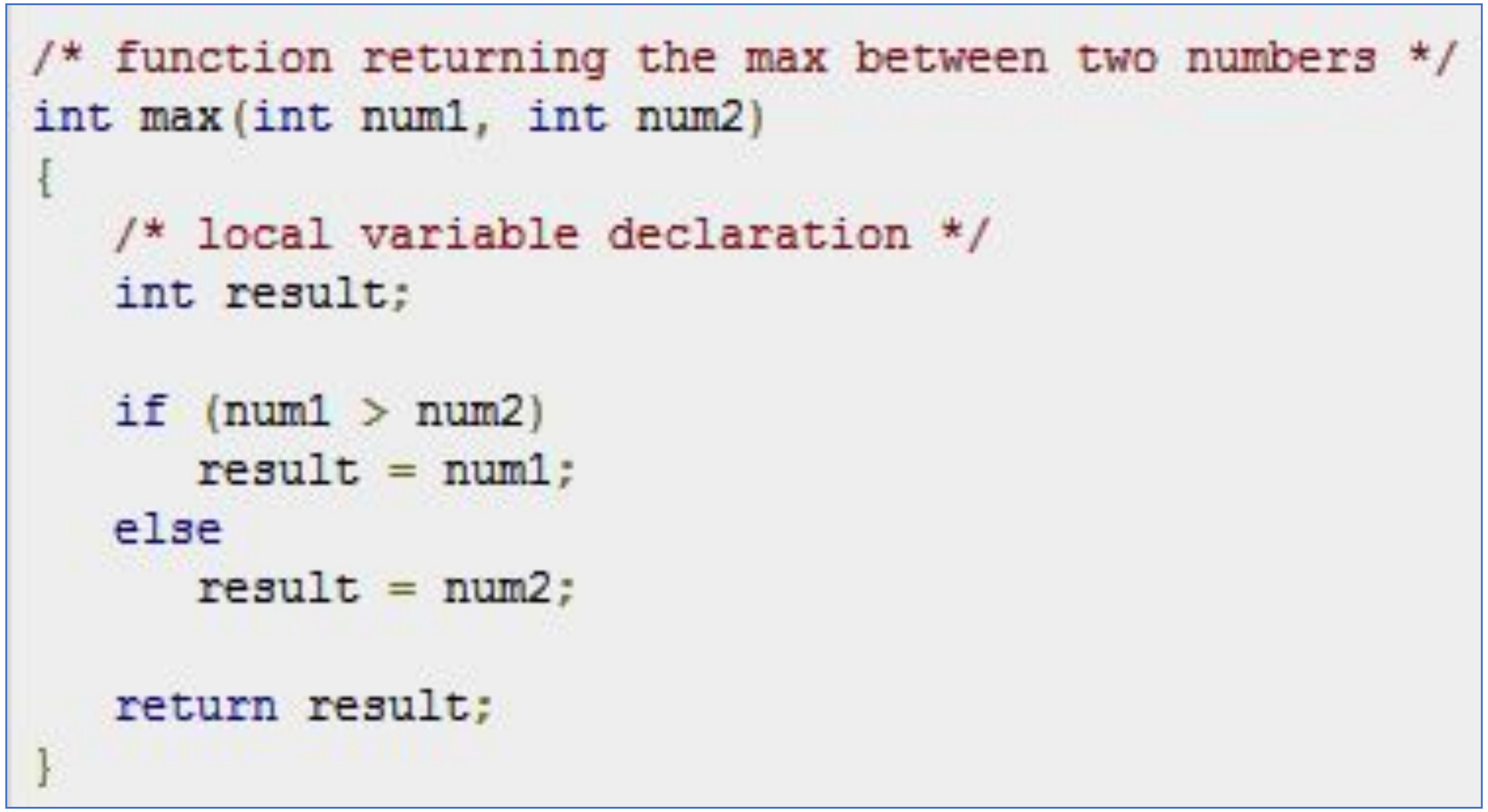
Course Teacher: Tanzina Afroz Rimi

Designation: Lecturer

Email: tanzinaafroz.cse@diu.edu.bd

Function

- Following is the source code for a function called **max()**. This function takes two parameters **num1** and **num2** and returns the maximum between the two:



```
/* function returning the max between two numbers */
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Function Declarations

- A function **declaration(function prototype)** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts:

```
return_type function_name( parameter list );
```

- For the above defined function *max*(), following is the function declaration:

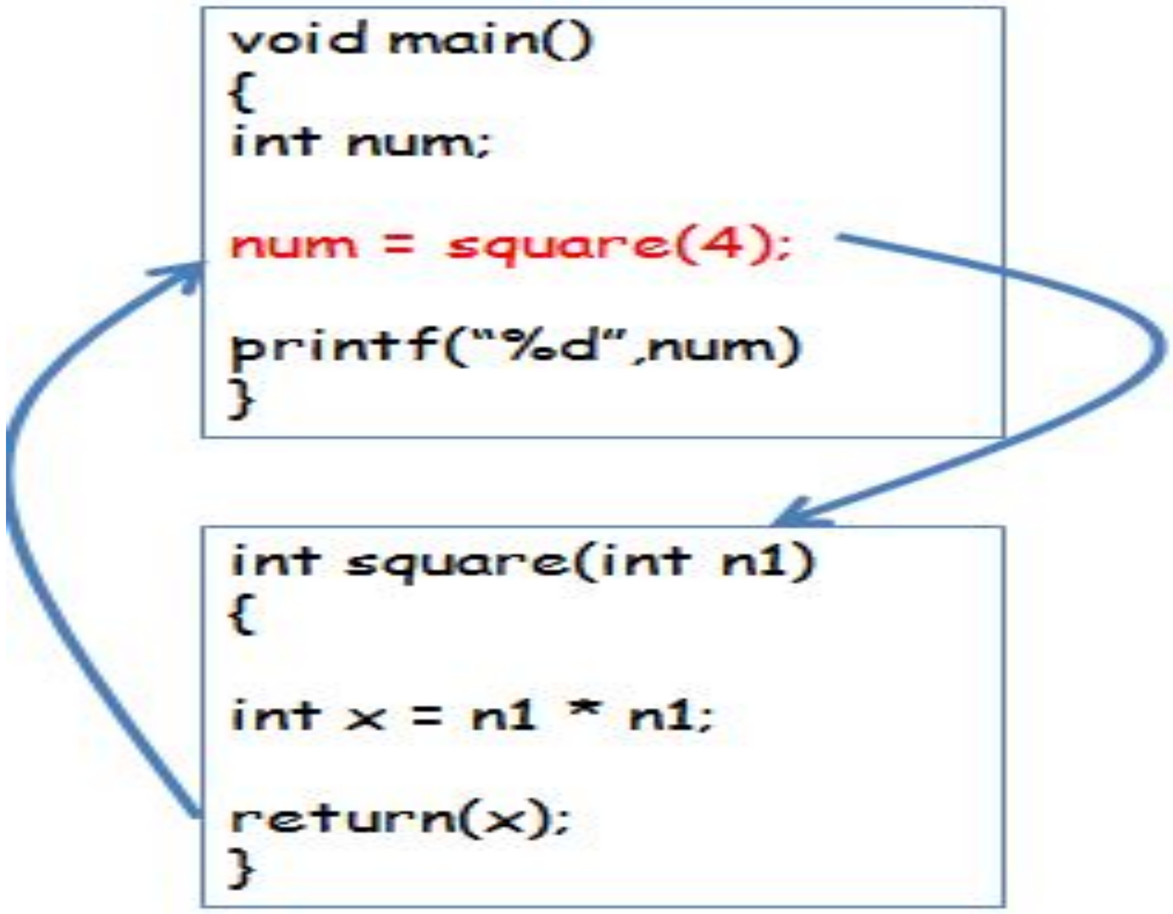
```
int max(int num1, int num2);
```

- Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

```
int max(int, int);
```

- Function declaration is required when you define a function in one source file and you call that function in another file. In such case you should declare the function at the top of the file calling the function.

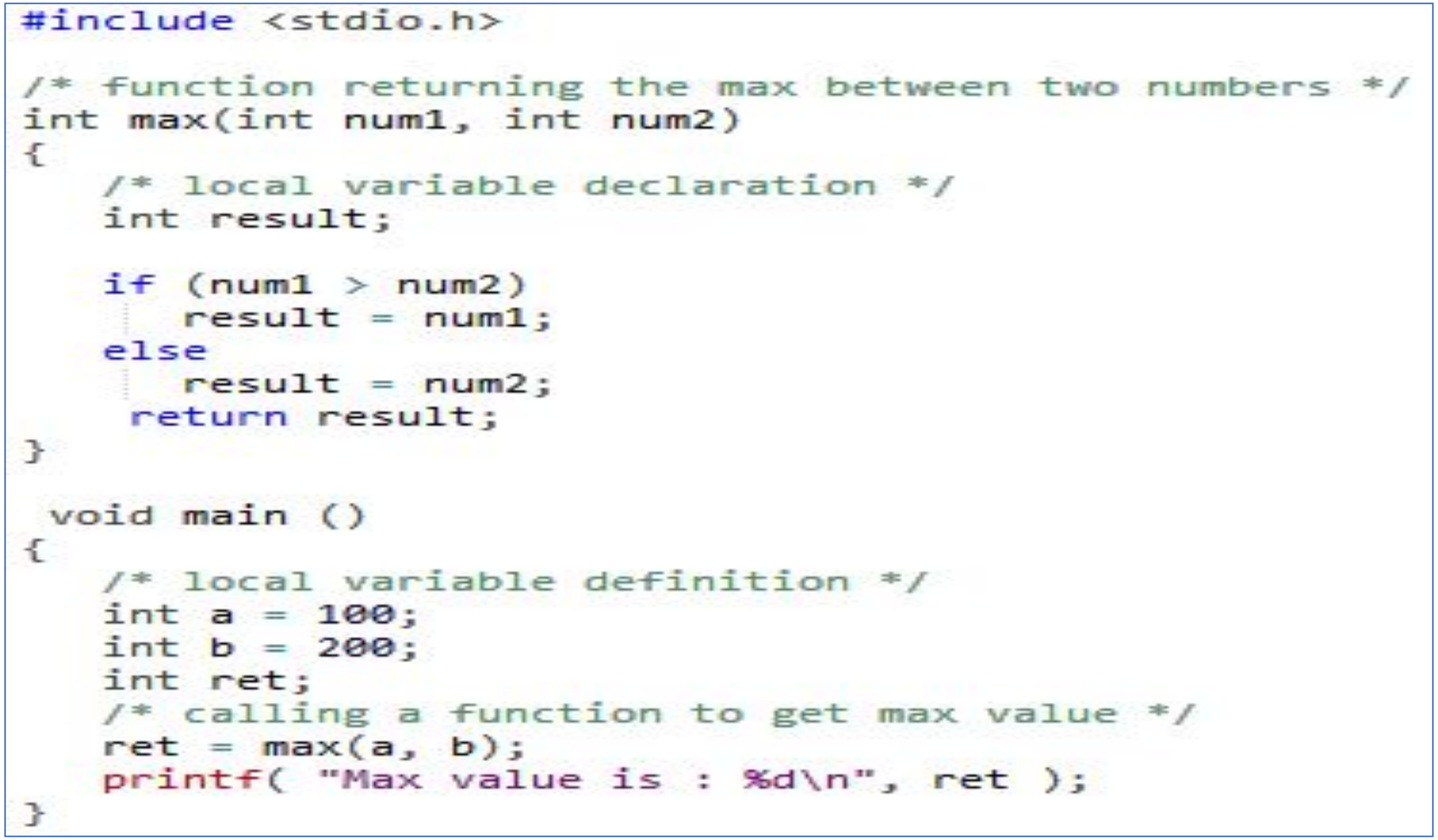
Calling a Function



```
void main()
{
int num;

num = square(4);

printf("%d",num)
}
```

```
int square(int n1)
{

int x = n1 * n1;

return(x);
}
```

Code Example: Calling a Function



```c
#include <stdio.h>

/* function returning the max between two numbers */
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;
    return result;
}

 void main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;
   /* calling a function to get max value */
   ret = max(a, b);
   printf( "Max value is : %d\n", ret );
}
```

RECURSIVELY DEFINED FUNCTIONS

- A function is said to be recursively defined if the function refers to itself such that
  1. There are certain arguments, called base values, for which the function does not refer to itself.
  2. Each time the function does refer to itself, the argument of the function must be closer to a base value.

Recursion

- RECURSION

  The process of defining an object in terms of smaller versions of itself is called recursion.
- A recursive definition has two parts:

1. BASE:

An initial simple definition which cannot be expressed in terms of smaller versions of itself.

2. RECURSION:

The part of definition which can be expressed in terms of smaller versions of itself.

THE FACTORIAL OF A POSITIVE INTEGER: Example

- For each positive integer n, the factorial of n denoted as n! is defined to be the product of all the integers from 1 to n:

$$n! = n.(n - 1).(n - 2) \dots 3 . 2 . 1$$

- Zero factorial is defined to be 1

$$0! = 1$$

EXAMPLE:

$0! = 1$	$1! = 1$
$2! = 2.1 = 2$	$3! = 3.2.1 = 6$
$4! = 4.3.2.1 = 24$	$5! = 5.4.3.2.1 = 120$
$! = 6.5.4.3.2.1= 720$	$7! = 7.6.5.4.3.2.1= 5040$

REMARK:

$$5! = 5 .4.3 . 2 . 1 \qquad = 5 .(4 . 3 . 2 .1) \qquad = 5 . 4!$$

In general, **$n! = n(n-1)!$** for each positive integer n.

THE FACTORIAL OF A POSITIVE INTEGER: Exam

o Thus, the recursive definition of factorial function F(n) is:

1. F(0) = 1
2. F(n) = n * F(n-1)

The Fibonacci numbers

- $f(0) = 0, f(1) = 1$
- $f(n) = f(n – 1) + f(n - 2)$

$f(0) = 0$

$f(1) = 1$

$f(2) = f(1) + f(0) = 1 + 0 = 1$

$f(3) = f(2) + f(1) = 1 + 1 = 2$

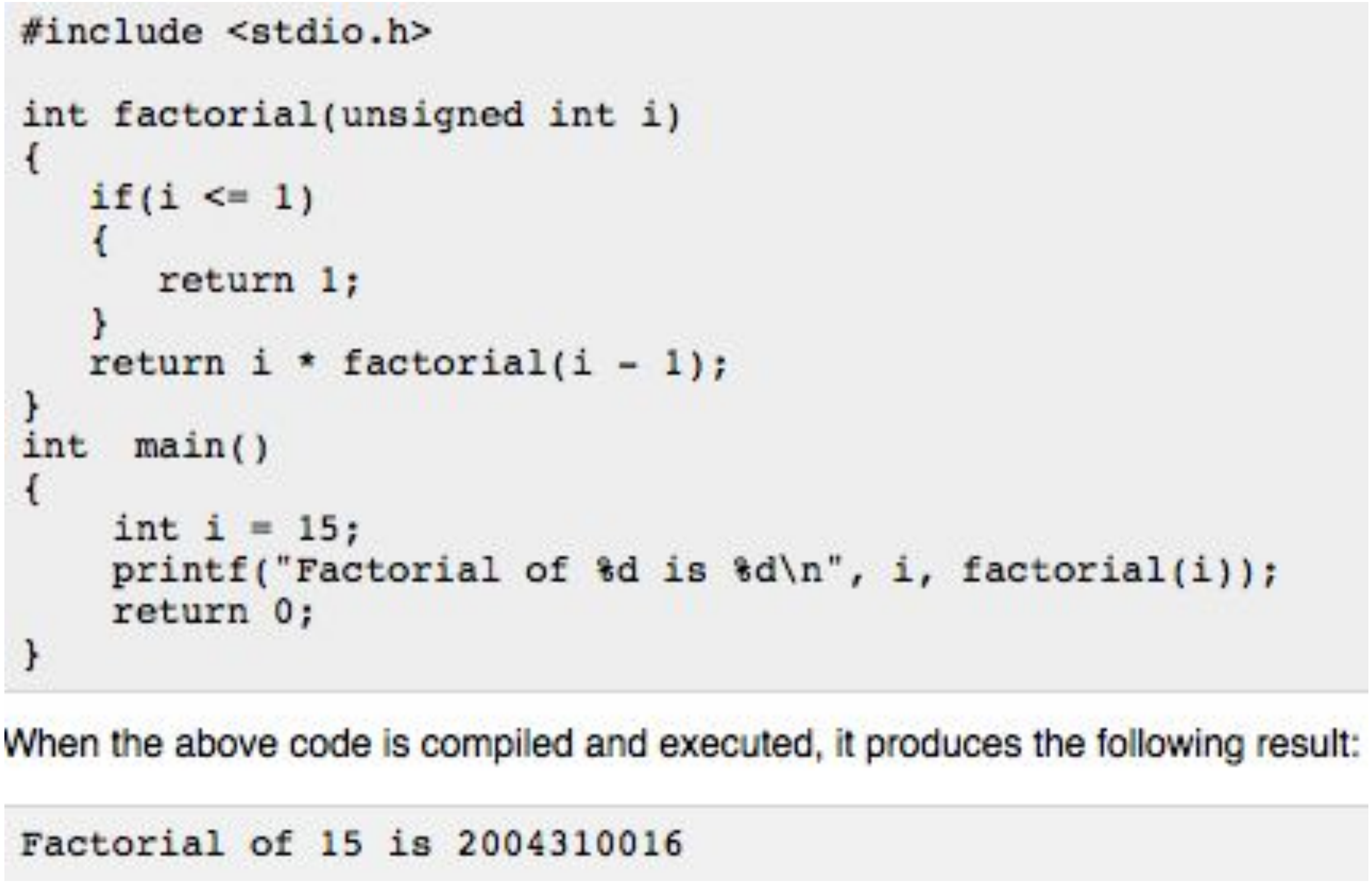
$f(4) = f(3) + f(2) = 2 + 1 = 3$

$f(5) = f(4) + f(3) = 3 + 2 = 5$

$f(6) = f(5) + f(4) = 5 + 3 = 8$

Number Factorial

- Following is an example, which calculates factorial for a given number using a recursive function:



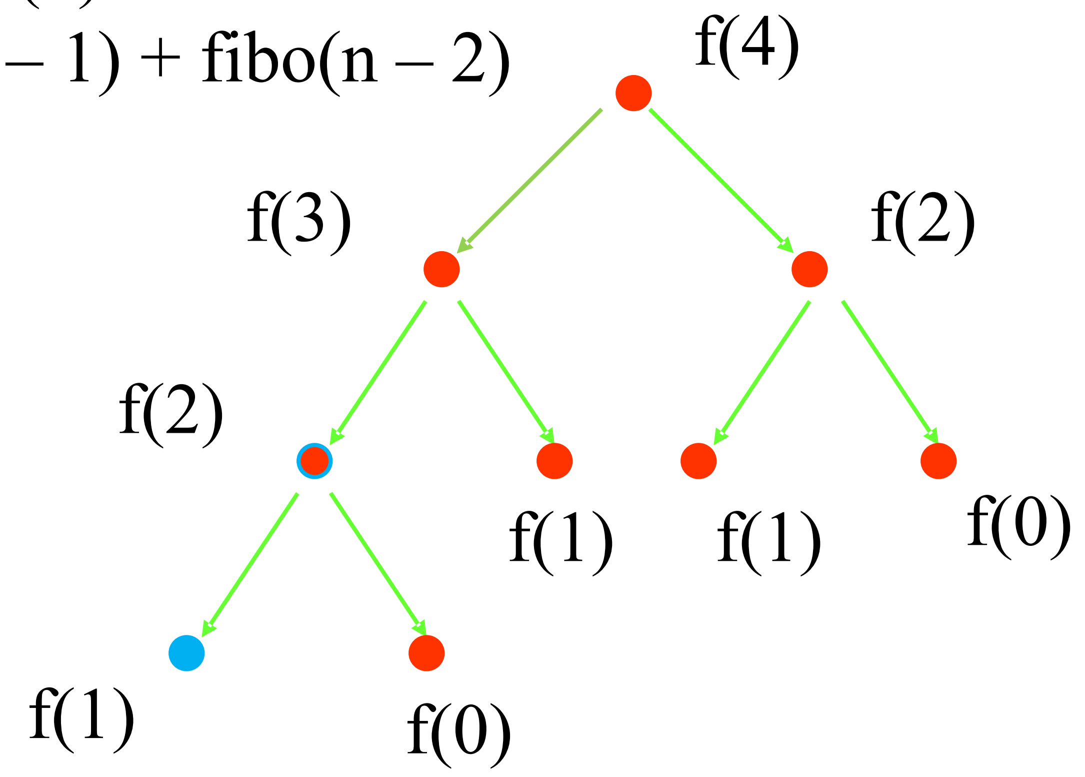
```c
#include <stdio.h>

int factorial(unsigned int i)
{
   if(i <= 1)
   {
      return 1;
   }
   return i * factorial(i - 1);
}
int  main()
{
    int i = 15;
    printf("Factorial of %d is %d\n", i, factorial(i));
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Factorial of 15 is 2004310016
```

RECURSIVELY DEFINED FUNCTIONS

```
procedure fibo(n: nonnegative integer)
    if n = 0 then fibo(0) := 0
    else if n = 1 then fibo(1) := 1
    else fibo(n) := fibo(n – 1) + fibo(n – 2)
end
```



USE OF RECURSION

- At first recursion may seem hard or impossible, may be magical at best. However, recursion often provides elegant, short algorithmic solutions to many problems in computer science and mathematics.
  - Examples where recursion is often used
    - math functions
    - number sequences
    - data structure definitions
    - data structure manipulations
    - language definitions

USE OF RECURSION

- For every recursive algorithm, there is an equivalent iterative algorithm.
- However, iterative algorithms are usually more efficient in their use of space and time.

EUCLID'S ALGORITHM

- Problem
  - Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n
  - **Examples:**

    gcd(60,24) = 12

    gcd(60,0) = 60

    gcd(0,0) = ?

EUCLID’S ALGORITHM

- Euclid’s algorithm is based on repeated application of equality

  gcd(m,n) = gcd(n, m mod n)

  until the second number becomes 0, which makes the problem trivial.

- Example:

  gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

TWO DESCRIPTIONS OF EUCLID’S ALGORITHM

- Step 1

  If n = 0, return m and stop; otherwise go to Step 2

- Step 2

  Divide m by n and assign the value fo the remainder to r

- Step 3

  Assign the value of n to m and the value of r to n. Go to Step 1.

```
while $n \neq 0$ {
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
}
return m
```

Textbooks & Web References

- Reference book iii (Chapter 3)
- Text Book (Chapter 31)
- Reference book ii (Chapter 1)
- www.geeksforgeeks.org
- www.topcoder.com

Thank you

&

Any question?